K[1] = “while\_\_“ bzw. 0x1004 \*(K +1)

K[2][4] = ‘o‘

\*(\*(K +2) + 4)

&K[2][4] = 0x100c +4= 0x1010

\*C[1] = ‘w‘

\*(\*(C + 1) + 0)

C[0] = “sizeof\_“ bzw. 0x100c

\*(C + 0)

C = 0x4000

pp\_c[0][2] = ‘z‘

\*(\*(pp\_c + 0) + 2)

pp\_c +4 = 0x4004

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2000 | 1000 | „do\_“ | 4000 | 100C | „sizeof\_“ | 1000 | do\_0 |
| 2004 | 1004 | „while\_“ | 4004 | 1004 | „while\_“ | 1004 | whil |
| 2008 | 100C | „sizeof\_“ | 4008 | 100C | „do\_“ | 1008 | e\_\_0 |
|  | … |  |  | … |  | 100C | size |
|  | 2000 | K |  | 4000 | C | 1010 | of\_0 |

while(n < 12){

n++;

x=Fkt(x,n);

}

for (n = 5; n < 12; n++){

x=Fkt(x, n);

}

switch (i){

case 3: ;

case 4: c1++; break;

default: c3++;

}

do{

a = a + 1;

}while( a < 20 );

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **N** | **Z** | **V** | **C** |  | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | r0=r1=r2  =r3=r4= | ee | 55 | ee | 55 |
| ldr | r0, =L1 | 0 | 0 | 0 | 0 | r0 = | 20 | 00 | 00 | 00 |
| mov | r1, #133 | 0 | 0 | 0 | 0 | r1 = | 00 | 00 | 00 | 85 |
| ands | r2, r1 | 0 | 0 | 0 | 0 | r2 = | 00 | 00 | 00 | 05 |
| ldrb | r3, [r0, #8]! | 0 | 0 | 0 | 0 | r3 = | 00 | 00 | 00 | 44 |
| movs | r4, r1, LSL #24 | 1 | 0 | 0 | 0 | r4 = | 85 | 00 | 00 | 00 |
| subs | r2, r1 | 1 | 0 | 0 | 0 | r2 = | ff | ff | ff | 80 |
| eor | r1, r3 | 1 | 0 | 0 | 0 | r1 = | 00 | 00 | 00 | c1 |
| ldr | r2, [r0, #4] | 1 | 0 | 0 | 0 | r2 = | 00 | 00 | 20 | 16 |
| ldr | r3, =L4 | 1 | 0 | 0 | 0 | r3 = | 20 | 00 | 00 | 14 |
| ldr | r1, [r0] | 1 | 0 | 0 | 0 | r1 = | 00 | 00 | 45 | 44 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **N** | **Z** | **V** | **C** |  | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | r0=r1=r2  =r3=r4= | aa | 55 | aa | 55 |
| ldr | r0, =L1 | 0 | 0 | 0 | 0 | r0 = | 20 | 00 | 00 | 38 |
| mov | r1, #0xa1 | 0 | 0 | 0 | 0 | r1 = | 00 | 00 | 00 | a1 |
| ands | r2, r1 | 0 | 0 | 0 | 0 | r2 = | 00 | 00 | 00 | 01 |
| ldrb | r3, [r0, #4]! | 0 | 0 | 0 | 0 | r3 = | 00 | 00 | 00 | 41 |
| movs | r4, r1, LSL #16 | 0 | 0 | 0 | 0 | r4 = | 00 | a1 | 00 | 00 |
| subs | r2, r1 | 1 | 0 | 0 | 0 | r2 = | ff | ff | ff | 60 |
| eor | r1, r3 | 1 | 0 | 0 | 0 | r1 = | 00 | 00 | 00 | e0 |
| ldr | r3, [r0, #8] | 1 | 0 | 0 | 0 | r3 = | 00 | 00 | 12 | 34 |
| adds | r3, r2, r3 | 1 | 0 | 0 | 1 | r3 = | 00 | 00 | 11 | 94 |
| ldrh | r1, [r0] | 1 | 0 | 0 | 0 | r1 = | 00 | 00 | 46 | 41 |

|  |
| --- |
| **Sprungegbefehle** |
| **unsigned** |
| bcs (carry set; borrow clear) |
| bhs (higher or same) |
| bhi (higher) |
| beq (equal) |
| bne (not equal) |
| bcc (carry clear) |
| blo (lower) “ “ |
| b ls (lower or same) |
| **signed** |
| blt (less than) |
| ble (less or equal) |
| beq (equal) |
| bne (not equal) |
| bge (greater or equal) |
| bgt (greater than) |

**ungerade Zahl**

ands r1, #0x01;lsb extrahieren,

bne LAB

**1111 X000**

and r2, #0xf7; 1111 X000

eors r2, #0xf0; 1111 0000 1sen invert.

, 0 hält ; Ergebnis dann 0x00

beq LAB

**XXXX 1111**

and r1, #0x0f; nibble extrahieren,

eors r1, #0x0f; invertieren

beq LAB

**XXXX 1111**

and r1, #0x0f; nibble extrahieren,

cmp r1, #0x0f; vergleichen

beq LAB

**11XX XX00**

and r2, #0xc3; 11XX XX00

eors r2, #0xc0; 1100 0000 1sen invert., 0 hält ; Ergebnis dann 0x00

beq LAB

|  |  |
| --- | --- |
| 0110 0111  +1101 1011  ----------  11111 111 C  ----------  0100 0010 | C = 1  V = 1 eor 1 = 0  V-los: C = 1 🡪 falsch (hier 1 🡪 falsch)  V-beh: V = 1 🡪 falsch (hier 0 🡪 richtig) |
| 0100 0000  (-0110 1011)  +1001 0101  -----------  00000 000 C  -----------  1101 0101 | C = 0  V = 0 eor 0 = 0  V-los: ¬C = 1 🡪 falsch (hier 1 🡪 falsch)  V-beh: V = 1 🡪 falsch (hier 0 🡪 richtig) |

Globale Variable: im Datenbereich mit 0 initialisiert

lokale Variable: in main()

|  |  |
| --- | --- |
| **FACTORIAL STACK** | |
| sp | r1 |
|  | r2 |
|  | r3 |
|  | r4 |
| fp | fp\_alt |
|  | lr |
|  | r0=X |
|  | r1=X |
| **POLY STACK** | |
| sp | r1 |
|  | r2 |
|  | r3 |
|  | r4 |
| fp | fp\_alt |
|  | lr |
|  | r0=X |
|  | r1=Y |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Adresse** | +0 | +1 | +2 | +3 |
| 0x2000 0000 | 0x24 | 0x18 | 0x12 | 0x17 |
| 0x2000 0004 | 0x33 | 0x34 | 0x20 | 0x43 |
| 0x2000 0008 | 0x44 | 0x45 | 0x00 | 0x00 |
| 0x2000 000C | 0x16 | 0x20 | 0x00 | 0x00 |
| 0x2000 0010 | 0x21 | 0x43 | 0x00 | 0x00 |
| 0x2000 0014 | 0x62 | 0x00 | 0x00 | 0x00 |

AREA MyDat, DATA

L1 DCB 0x24, 24, 0x12 , 0x17

L2 DCB "34 CDE"

DCD 0x2016

L3 DCD 0x4321, ‘C’

L4 EQU L3+4

ALIGN 4

L1 = 0x2000 0000

L2 = 0x2000 0004

L3 = 0x2000 0010

L4 = 0x2000 0018

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Adresse** | +0 | +1 | +2 | +3 |
| 0x2000 0038 | 0x21 | 0x81 | 0x12 | 0x00 |
| 0x2000 003c | 0x41 | 0x46 | 0x46 | 0x45 |
| 0x2000 0040 | 0x00 | 0x00 | 0x00 | 0x00 |
| 0x2000 0044 | 0x34 | 0x12 | 0x00 | 0x00 |
| 0x2000 0048 | 0x65 | 0x00 | 0x00 | 0x00 |
| 0x2000 004c | 0x62 | 0x00 | 0x00 | 0x00 |

AREA MyDat, DATA

L1 DCB 0x21, 129, 0x12

ALIGN 4

L2 DCB "AFFE",0

ALIGN 4

L3 DCD 0x1234, ‘e’, ‘b’

L4 EQU L3+4

L1 = 0x2000 0038

L2 = 0x2000 003c

L3 = 0x2000 0044

L4 = 0x2000 0048

struct Student {

char Name[25];

long int MatrikelNr;

char Studiengang[3];

} Std1, Std2, Std3;

struct Student Stud1 = {"Zuse", 125716, "TI"};

struct Student Stud2 = {"Gates", 125609, "AI"};

Student Std[20];

strcpy( Std[0].Name, "Winter");

AREA MyData, DATA, align = 2

GLOBAL Ergebnis\_2

text DCB "RMP SS16",0

COEF DCD 0x3, 0x02, 0x1; a, b , c

X DCD 2

Y DCD 3

Ergebnis\_2 SPACE 32

; …

;--------------------------------------------

; main subroutine

;--------------------------------------------

EXPORT main [CODE]

main PROC

; Vorbereitung POLY mit Parameter per Stack

ldr r2,=X;

ldr r0,[r2];

ldr r2,=Y;

ldr r1,[r2];

push{r0, r1}; X in r0 oberhalb von Y in r1 auf Stack

bl POLY

add sp,#8; Korrektur für 2 Variablen auf Stack

ldr r3,=Ergebnis\_2;

str r0,[r3, #4]

forever b forever ; nowhere to retun if main ends

ENDP

POLY PROC

push{lr, fp}; fp = r11 , lr = r14

mov fp, sp; sp = r13

push{r1,r2, r3, r4};

ldr r1, [fp, #8]; X in r1

ldr r2, [fp, #12]; Y in r2

;z = y\*(a\*X + c) + b\*x

ldr r3,=COEF;

ldr r4,[r3],#8; a

mul r0, r4, r1; aX

ldr r4,[r3],#-4; c

add r0, r0, r4; aX+c

mul r0, r0, r2; Y(aX+c)

ldr r4,[r3]; b

mul r4, r1, r4; bX

add r0, r0, r4; z

pop{r1 - r4}

pop{lr, fp}

bx lr

;

ENDP

ALIGN

END

; Vorbereitung FACTORIAL mit Parameter per Stack

ldr r2,=X;

ldr r0,[r2];

ldr r1,[r2];

push{r0, r1}; X in r0 oberhalb von r1 auf Stack

bl FACTORIAL

add sp,#8; Korrektur für 2 Variablen auf Stack

ldr r3,=Ergebnis\_2;

str r0,[r3, #4]

; ….

ENDP

FACTORIAL push{lr, fp}; fp = r11 , lr = r14

mov fp, sp; sp = r13

push{r1,r2, r3, r4};

ldr r1, [fp, #8]; X in r1

;ldr r2, [fp, #12]; X in r2

mov r0,#0x01 ; Ergebnis

mov r2,#0x00 ; cnt

WHILE cmp r1,r2;

beq RET

add r2, r2, #0x01;

mul r0, r0, r2; Rueckgabe mit r0 lr

b WHILE

RET pop{r1 - r4}

pop{lr, fp}

bx lr

;

ALIGN

END